An Optimal Algorithm for Finding Clique-Cross Partitions*

Hazel Everett[†] Sulamita Klein[‡] Bruce Reed[§]

Abstract

We consider the problem of determining whether or not the vertices of a graph can be partitioned into four disjoint cliques A, B, C, D such that vertices of A are not adjacent to vertices of C and vertices of B are not adjacent to vertices of D. We call this a clique-cross partition. We give an O($m + n$)-time algorithm to determine if a graph G with n vertices and m edges has such a partition, and give the partition in the case it exists.

1 Introduction

A *clique-cross partition* (CCP) of a graph G is a partition of the vertices of G into four disjoint cliques A, B, C, D such that :

- if $x \in A$ and $y \in C$, then $\{x, y\} \notin E$, and
- if $x \in B$ and $y \in D$, then $\{x, y\} \notin E$.

Observe that there are no restrictions on the edges between vertices in A and B, B and C, C and D and D and A.

We can represent a graph G that has a CCP by the diagram in Figure 1 above. The dashed line between A and C represents the property that no vertex of A is adjacent to any vertex of C. Similarly for B and D. The circles represent cliques-sets of pairwise adjacent vertices.

The purpose of this paper is to present a linear time algorithm to determine whether a graph G has a clique-cross partition and to find such a partition, if it exists, in time proportional to the size of the graph. We consider only finite

*This research was partially supported by NSERC, FCAR, CNPq and PRONEX 107/97.

[†]Departement d'informatique, Université du Québec à Montréal, Montréal, Québec, H3C 3P8, Canada.

[‡]Departamento da Ciência da Computação-I.M., COPPE/Sistemas, Universidade Federal do Rio de Janeiro, RJ, 21945-970, Brasil.

[§]C.N.R.S., Université Pierre et Marie Curie, Paris, France.

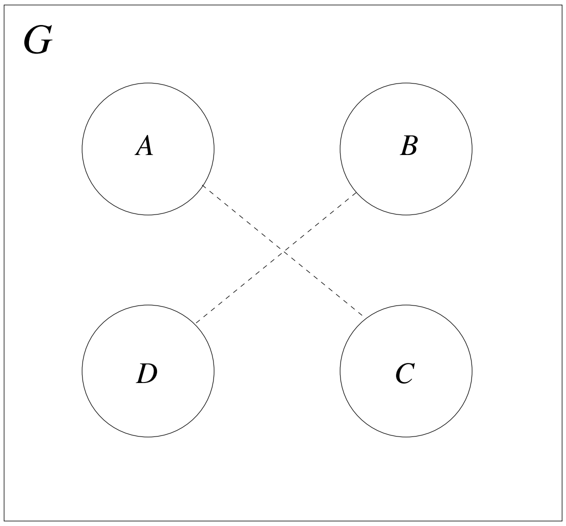

Figure 1: *CCP of graph G. A, B, C and D are cliques.*

undirected graphs $G = (V, E)$ with no loops or multiple edges, where V is the set of vertices of G and E is the set of edges of G.

The clique-cross partition problem is a particular case of the more general clique covering problem:

Given a graph H with $\mid V(H) \mid = k$, is there a partition $V_1, V_2, ..., V_k$ of the vertices of G such that V_i is a clique, $1 \leq i \leq k$, and the graph $\hat{G}$ obtained by contracting each set V_i to a single vertex v_i is a partial subgraph of H ? The graph $\hat{G} = (\hat{V}, \hat{E})$ is defined by: $\hat{V} = \bigcup\{v_i\}\ i = 1, \ldots, k$ and $\hat{E} = \{\{v_i, v_j\} \mid$ there is an edge between a vertex of V_i and a vertex of $V_j\}$.

MacGillivray and Yu [4] showed that if H is a triangle-free graph, then the clique-covering problem can be solved in O(n^{k+2})-time. In particular, if we take H to be the chordless cycle on 4 vertices, their algorithm solves the clique-cross partition problem in O(n^6)-time.

It is interesting to note that the problem of determining if a graph admits a CCP is equivalent to the problem of determining whether a graph can be covered by two complete bipartite graphs (*A graph G has a CCP if and only if its complement $\overline{G}$ has a covering by two complete bipartite graphs*). The problem of covering a graph by three complete bipartite graphs is NP-complete [4].

In this paper we show how our problem can be transformed into an instance of 2-SAT. Let $X = \{x_1, x_2, ..., x_m\}$ be a set of boolean variables. A truth assignment for X is a function $t : X \rightarrow \{true, false\}$. Let C be a boolean formula that is a conjunction of clauses , where each clause is a disjunction of literals, and each literal is either a variable x_i or the negation of a variable, $\overline{x_i}$. A formula C over X is satisfiable if and only if there exists some truth assignment for X that satisfies C. Such a truth assignment is called a satisfying truth assignment for C. The *satisfiability problem* is to determine if there is a satisfying truth assignment for C. It is well known that the satisfiability problem is NP-complete for formulas with three literals per clause (see [3]). However, the satisfiability problem for formulas with two literals per clause (2-SAT) is solvable in polynomial time [3]; Even,Itai, and Shamir [2] outlined the first linear time algorithm for 2-SAT, and later, Aspvall, Plass and Tarjan [1] presented a simpler linear time algorithm which they called the *two-satisfiability algorithm*.

2 The Algorithm

We shall describe a linear time algorithm which given two vertices a and b determines if G has a CCP such that $a \in A$ and $b \in B$. We could apply this algorithm to all pairs of vertices of G to determine if G has a CCP but this would take too long. Instead we will first find three vertices x, y, z such that if G has a CCP then G has either a CCP with $x \in A$, $y \in B$ or G has a CCP with $y \in A$ and $z \in B$.

Next we partition the vertex set $V^* = V \setminus \{a, b\}$ of G into the following four sets:

$$
\begin{aligned}
AB &= \{x \in V^* \mid \{x,a\} \in E \text{ and } \{x,b\} \in E\} \\
BC &= \{x \in V^* \mid \{x,a\} \notin E \text{ and } \{x,b\} \in E\} \\
CD &= \{x \in V^* \mid \{x,a\} \notin E \text{ and } \{x,b\} \notin E\} \\
DA &= \{x \in V^* \mid \{x,a\} \in E \text{ and } \{x,b\} \notin E\}.
\end{aligned}
$$

Consider a vertex $x \in AB$. Since x is adjacent to $a \in A$ it cannot be in clique C and since x is adjacent to $b \in B$ it cannot be in D. In other words x must be assigned to either A or B. Thus we have:

$$
\begin{aligned}
AB &\subseteq \{x \in V^* \mid x \in A \text{ or } x \in B\} \\
BC &\subseteq \{x \in V^* \mid x \in B \text{ or } x \in C\} \\
CD &\subseteq \{x \in V^* \mid x \in C \text{ or } x \in D\} \\
DA &\subseteq \{x \in V^* \mid x \in D \text{ or } x \in A\}.
\end{aligned}
$$

It is easy to see that G contains a CCP with $a \in A$ and $b \in B$ if and only if the following conditions are satisfied:

(I) if the vertices x and y are not adjacent and they belong to the same set (AB, BC, CD or DA), then they are assigned to different cliques.

(II) if x and y are adjacent and $x \in AB$, $y \in CD$ then if x is assigned to A then y is assigned to D and if x is assigned to B, y is assigned to C. Similarly for $x \in DA$ and $y \in BC$.

(III) if x and y are not adjacent and $x \in AB$, $y \in DA$ then if x is assigned to A, y is assigned to D. Similarly for $x \in BC$ and $y \in AB$, $x \in CD$ and $y \in BC$, and $x \in DA$ and $y \in CD$.

(IV) if x and y are adjacent and $x \in AB$, $y \in DA$ then if y is assigned to D, x is assigned to A. Similarly for $x \in BC$ and $y \in AB$, $x \in CD$ and $y \in BC$, and $x \in DA$ and $y \in CD$.

Now for a given pair a,b we construct an instance of 2-SAT where each vertex of G corresponds to a variable and each instance of rules I-IV corresponds either to one or two clauses. Then we can apply the two-satisfiability algorithm [1] to solve this problem in O($m + n$)-time.

(CCP-algorithm)
Input: a graph $G = (V, E)$
Output: *YES-G has a CCP* or *NO-G does not have a CCP*. If the answer is *YES*, the algorithm also returns a partition of V into cliques A, B, C and D that form a CCP.

Step 0 Carry out the initialization procedure given in Section 3. This procedure either returns *NO-G does not have a CCP* or it returns *YES-G has a CCP* and also the four cliques A, B, C, D, or it returns an ordered triple of vertices of G. In either of the first two cases we are finished. Otherwise, let the first vertex of this triple be a, the second b, assume $a \in A$ and $b \in B$, and proceed to the next step.

Step 1 Partition the set $V^* = V \setminus \{a, b\}$ into the four sets AB, BC, CD, DA:
$AB = \{x \in V^* \mid \{x, a\} \in E$ and $\{x, b\} \in E\}$
$BC = \{x \in V^* \mid \{x, a\} \notin E$ and $\{x, b\} \in E\}$
$CD = \{x \in V^* \mid \{x, a\} \notin E$ and $\{x, b\} \notin E\}$
$DA = \{x \in V^* \mid \{x, a\} \in E$ and $\{x, b\} \notin E\}$

Step 2 Construct an instance of 2-SAT as follows. We have one variable $moves_x$ for each vertex of G. The idea is to transform the sets AB, BC, CD and DA into the sets A, B, C and D, respectively, by moving vertices. Consider, for example, a vertex $x \in AB$. Vertex x is either in set A or in set B. If it turns out to be in B then variable $moves_x$ is going to be assigned the value $true$, otherwise (i.e. if $x \in A$) it is going to be assigned the value $false$. We construct clauses corresponding to conditions (I)-(IV) given above as follows:

(i) for each $\{x, y\} \notin E$ and x and y belonging to the same set AB, BC, CD or DA we have the clauses:

$$(moves_x \vee moves_y) \wedge (\overline{moves_x} \vee \overline{moves_y})$$

(ii) for each $\{x, y\} \in E$, $x \in AB$, $y \in CD$ we have the clauses:

$$(moves_x \vee moves_y) \wedge (\overline{moves_x} \vee \overline{moves_y})$$

Similarly for $\{x, y\} \in E$, $x \in DA$, $y \in BC$.

(iii) for each $\{x, y\} \notin E$, $x \in AB$ and $y \in DA$ we have:

$$moves_x \vee \overline{moves_y}$$

Similarly for $x \in BC$ and $y \in AB$, $x \in CD$ and $y \in BC$, and $x \in DA$ and $y \in CD$.

(iv) for each $\{x, y\} \in E$, $x \in AB$ and $y \in DA$ we have:

$$\overline{moves_x} \vee moves_y$$

Similarly for $x \in BC$ and $y \in AB$, $x \in CD$ and $y \in BC$, and $x \in DA$ and $y \in CD$.

Let C be the conjunction of these clauses.

Step 3 Apply the two-satisfiability algorithm [1] to C.

Step 4 If C is unsatisfiable then G does not have a CCP for this chosen pair of vertices, so proceed to step 5. Otherwise set:

$A = \{x \in AB \mid moves_x = false\} \bigcup \{x \in DA \mid moves_x = true\} \bigcup \{a\}$

$B = \{x \in BC \mid moves_x = false\} \bigcup \{x \in AB \mid moves_x = true\} \bigcup \{b\}$

$C = \{x \in CD \mid moves_x = false\} \bigcup \{x \in BC \mid moves_x = true\}$

$D = \{x \in DA \mid moves_x = false\} \bigcup \{x \in CD \mid moves_x = true\}$

Return *YES-G has a CCP* and the sets A, B, C, D and stop.

Step 5 Repeat steps 1 to 4 with a the second vertex of the ordered triple given in Step 0 and b the third vertex. If no CCP is found return *NO-G does not have a CCP* and stop.

3 The Initialization Procedure

Recall that the purpose of this procedure is to find three vertices of G such that G has a CCP if and only if it has a CCP with one of these vertices in A and one in B. In the case that no such three vertices are found, the procedure correctly determines whether or not G has a CCP.

We start by counting the edges of G. If $m < 4\binom{\lfloor n/4 \rfloor}{2}$ then clearly G cannot be partitioned into 4 cliques. Return *NO-G does not have a CCP* and stop. Otherwise we find the connected components of G. Note that if G has a CCP then each connected component contains at least one of the four cliques. If G has more than 4 components then return *NO-G does not have a CCP* and stop. Otherwise we look for a P_3: an induced path with three vertices x, y, z and two edges $\{x, y\}$ and $\{y, z\}$. Observe that if G has a P_3 then either x, y and z are all in different cliques or x and y are in the same clique and z is in a different clique. In any case we can assume without loss of generality that either $x \in A$ and $y \in B$ or $y \in A$ and $z \in B$. If there is a P_3 then return the vertices of any such P_3. Otherwise let G_1, G_2, G_3, G_4 be the connected components of G where G_2, G_3 and G_4 are possibly empty, return *YES-G has a CCP with* $A = V(G_1)$, $B = V(G_2)$, $C = V(G_3)$, $D = V(G_4)$ and stop.

4 The Theorem

We are now ready to state our main theorem:

Theorem 1 *Determining whether a graph has a CCP can be accomplished in O($m + n$)- time.*

Proof: It is clear that G has a CCP if and only if there exists a satisfying truth assignment for C. We can estimate the running time of the CCP-algorithm as follows. The initialization procedure, Step 0, can be done in O($m + n$) time. Indeed, finding the connected components of G is linear in the size of the graph [5]. Determining if it has a P_3 can be done in time linear in the size of the graph using a breath first search. In Step 1 it is clear that the construction of each of the four sets takes time proportional to n. In Step 2 the construction of an instance of 2-SAT takes O(n^2). Step 3, the two-satisfiability algorithm, can be done in linear time in the number of clauses [1], which in this case is O(n^2). Clearly Step 4 can be done in linear time. Then Step 1 to Step 4 takes O(n^2) time. According to Step 5 we will apply steps 1 to 4 at most twice. So the algorithm takes O(n^2) time. But we note that in the beginning of the initialization procedure we counted the edges of G and continued only if m is greater than or equal to $4\binom{\lfloor n/4 \rfloor}{2}$. Thus we execute steps 1 to 4 only if n^2 is in O(m). $\square$

Acknowledgements: We would like to thank Celina M. H. de Figueiredo and Marcia R. Cerioli for careful readings of earlier drafts of this paper and René Ferland for giving invaluable advice on LaTeX.

References

[1] B.Aspvall, F.Plass, R.E.Tarjan, "A linear-time algorithm for testing the truth of certain quantified boolean formulas", *Information Processing Letters*,

8(3) (1979) 121-123.

[2] S. Even, A. Itai and A. Shamir, "On the complexity of timetable and multi-commodity flow problems", *SIAM J.Comput.* 5(4) (1976) 691-703.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco, (1979).

[4] G.MacGillivray, M.L.Yu, "Generalized covers of graphs", manuscript.

[5] R.E.Tarjan, "Depth First Search and Linear Graph Algorithms", *SIAM J.Comput.* 1(2) (1972) 146-160.